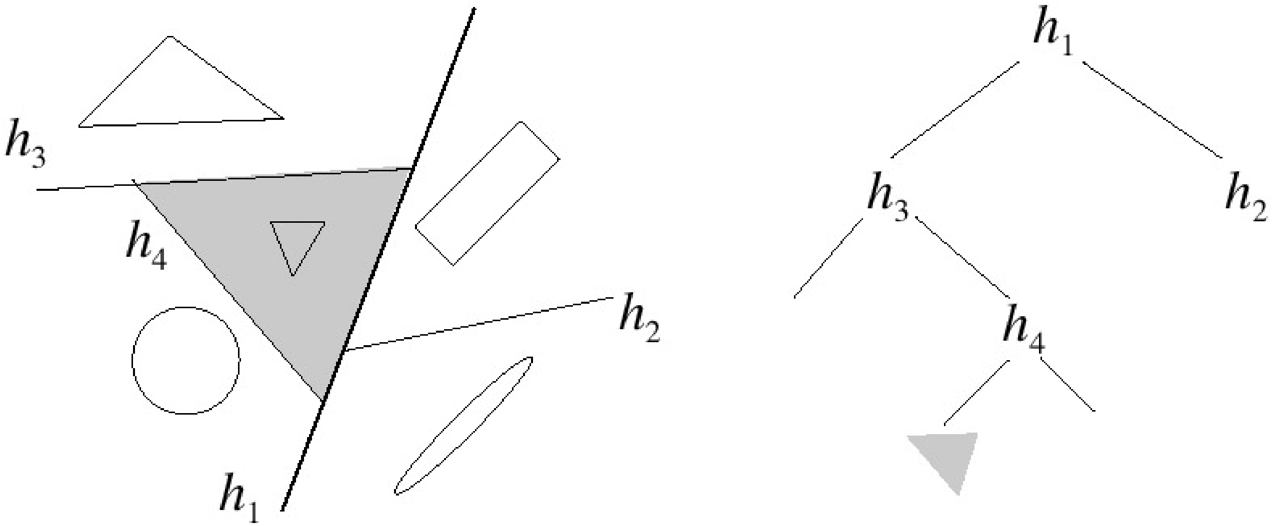
Geometric Structures

3. BSP

Martin Samuelčík
samuelcik@sccg.sk, www.sccg.sk/~samuelcik, I4

BSP trees

- Binary Space Partitioning
- Generaliyation of k-d trees, partitioning of space using arbitrary hyperplanes
- Enabling sorting of objects
- Doom, Quake, Half-life...



BSP tree

- Let *S* is set of objects (points, polygons,...)
- *S(v)* is set of objects for BSP tree node *v*
- BSP tree *T(S)* for set *S* is defined:
  - If $|S| <= 1$, then *T(S)* is leaf containing *S*
  - If $|S| > 1$, then *v* is root *T* and *v* contains divider hyperplane h_v, set $S(v)=\{x \in S, x \in h_v\}$ and two sibling nodes(subtrees) for objects on left respectively right side of hyperplane h_v

$$S^- := \{x \cap h_v^- | x \in S\}$$
$$S^+ := \{x \cap h_v^+ | x \in S\}$$

BSP tree creation

```
struct HyperPlane
{
        vector<float> coefficients;
}
```

```
struct BSPTreeNode
{
        List polygons;
        HyperPlane partition;
        BSPTreeNode* front;
        BSPTreeNode* back;
}
```

```
struct BSPTree
{
        BSPTreeNode* root;
}
```

```
BSPTree* BuildBSPTree(List polygons)
{
        result = new BSPTree;
        result->root = BuildBSPTreeNode(polygons);
        return result;
}
```

```
BSPTreeNode*  BuildBSPTreeNode (list polygons)
{
        if (polygons.IsEmpty ()) return NULL;
        BSPTreeNode* tree = new BSPTreeNode;
        polygon* root = polygons.GetFromList ();
        tree->partition = root->GetHyperPlane ();
        tree->polygons.AddToList (root);
        list front_list, back_list;polygon* poly;
        while ((poly = polygons.GetFromList ()) != 0)
        {
                int result = tree->partition.ClassifyPolygon (poly);
                switch (result)
                {
                        case COINCIDENT:
                          tree->polygons.AddToList (poly);
                          break;
                        case IN_BACK_OF:
                          back_list.AddToList (poly);
                          break;
                        case IN_FRONT_OF:
                          front_list.AddToList (poly);
                          break;
                        case SPANNING:
                          polygon   *front_piece, *back_piece;
                          SplitPolygon (poly, tree->partition, front_piece, back_piece);
                          back_list.AddToList (back_piece);
                          front_list.AddToList (front_piece);
                          break;
                }
        }
        tree->front = BuildBSPTreeNode (front_list);
        tree->back = BuildBSPTreeNode (back_list);
}
```

Hyperlpanes

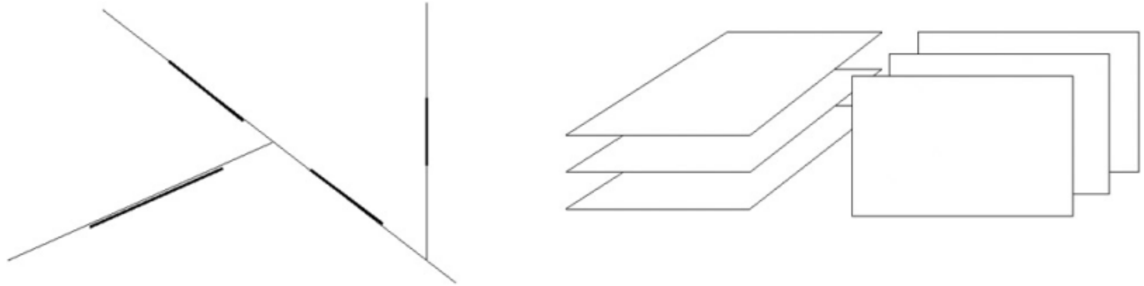
- Line, plane, ...
- Implicit representation for *d*-dimensional space: $a_1.x_1+a_2.x_2+\dots a_d.x_d+a_{d+1}=0$
- $(a_1,a_2,\dots a_d)$ – normal, representing also orientation of hyperplane, defining inside or outside part
- Point test – sign od result after computation of implicit representation with point coordinates
- Polygon test – comparing point test signs for each vertex of polygon
- Splitting polygons– searching for intersection of boundary segments with hyperplane

BSP tree splitting techniques

- Auto-configuration – $O(n^2)$
- Arbitrary splitting techniques, time complexity computation: $$T(n) = n + 2T(\frac{n}{2} + \alpha n) \in O(n^{1+\delta})\ ,\ 0 < \alpha < \frac{n}{2},$$
  - α = average count of polygons split in nodes

α	0.05	0.2	0.4
	$n^{1.15}$	n^2	n^7

- For each polygon, choose point-representative (barycenter, center of BB, ...) and find hyperplane, that splits set of representatives into two subsets with same count



Cost heuristics for split

- Computing quality cost of split
- Tree cost $$C(T) = 1 + P(T^-)C(T^-) + P(T^+)C(T^+),$$
  - C – cost function, P – probability of visiting tree
  - For example for point location(inside or outside of object) $P(T^-) = Vol(T^-)/Vol(T)$, for raytracing area of cell bounding subtree
- Local heuristics
  - S – number of polygons, objects, s – split objects count $$C(T) = 1 + |S^-|^\alpha + |S^+|^\alpha + \beta s,$$

Automatic subdivision

- Hyperplane defined by one of given polygons
- Choose large polygons
  - Large polygons have higher probability to be split, so this way remove it sooner from set of polygons
  - For first k largest polygons, compute cost function C(T) and choose polygon with lowest cost
- Random choose k polygons
  - From k polygons, choose one that will create smallest count of fragments
- Used constants for cost function computation
  - $\alpha = 0.8, \ldots, 0.95$; $\beta = 1/4, \ldots, 3/4$
  - $k = 5$

BSP tree for raytracing

- Organizing tree based on specifics of geometric search – for example rays emit from one point
- Cost of queries

$$C(\text{query}) = \#\,\text{nodes visited} \leq \text{depth(BSP)} \cdot \#\,\text{stabbed leaf cells.}$$

- We want to hit as less nodes as possible, polygons with higher hit probability are places in higher in tree hierarchy
- Probability of ray-polygon intersection:
  - If the angle of ray direction and polygon normal is smaller, probability is higher
  - If the polygon is larger, probability id higher

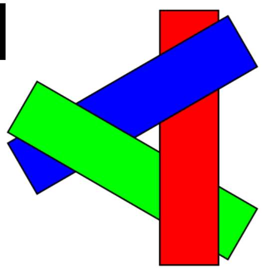
$$\text{score}(p) = \int_D w(S, p, l)\omega(l)dl, \qquad w(S, p, l) = \sin^2(\mathbf{n}_p, \mathbf{r}_l)\frac{\text{Area}(p)}{\text{Area}(S)},$$

Self-organizing BSP trees

- If distribution of polygons is not known or cost function is harder to compute
- Constructing only necessary parts of BSP tree
- Each node also holds info about currently unused polygons, that were not used until now
- Remembering how many times node of tree was visited, if counter is above limit, the node is subdivided and new subtree of node created
- Computing also intersection count of ray and polygons in unsplit node, this counter is later used for choosing split hyperplane

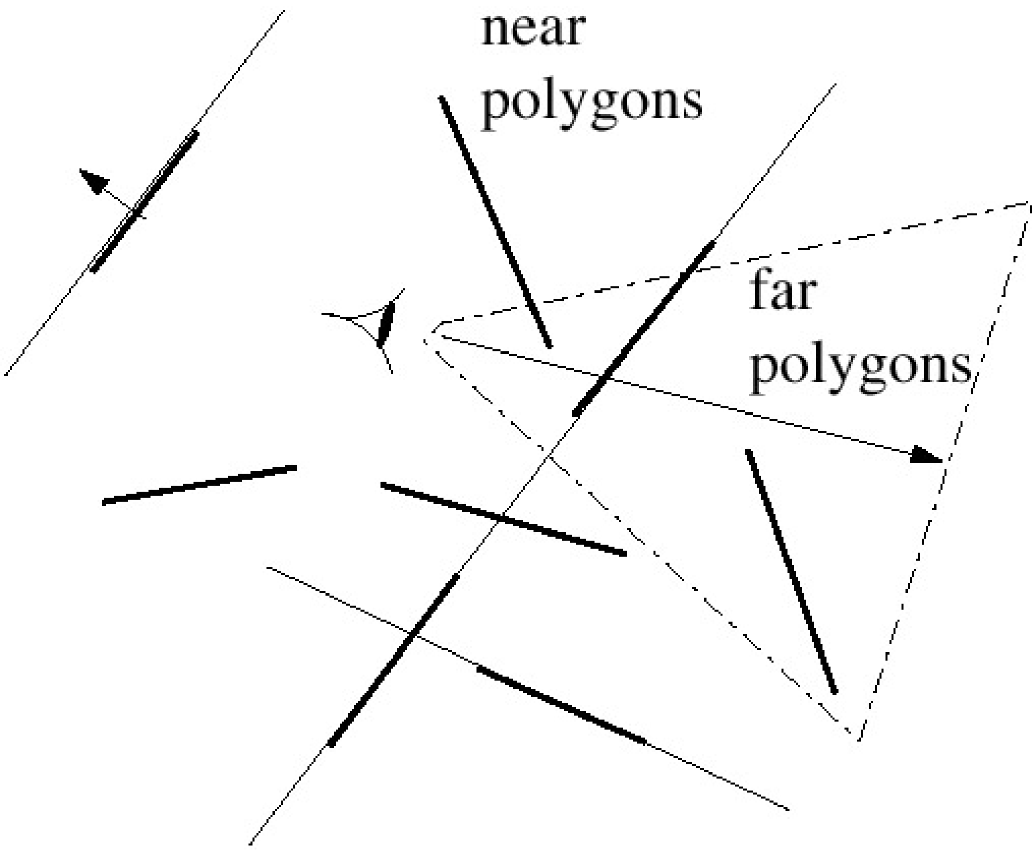
Visibility determination

- Determine occluded parts of polygons in 3D scene
- Painter algorithm – painting from background towards front (polygons must be in simple positions)



- BSP – having partition of space, each hyperplane in node splits space into two halves, half-space where camera is positioned contains objects nearer to camera, other half-space contains objects far from camera
- Always comparing split hyperplane with camera position

Visibility determinantion



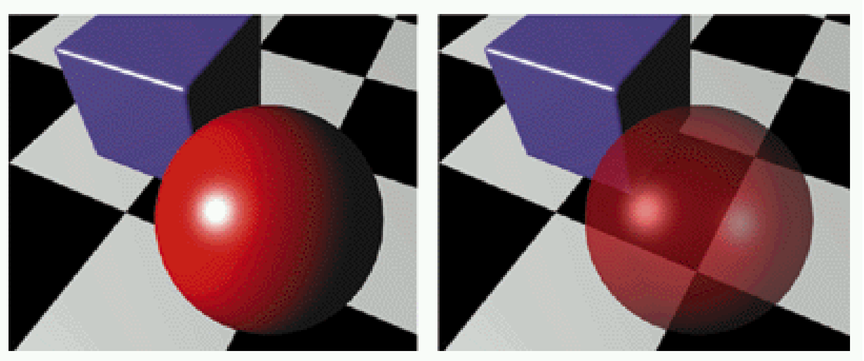

```
void DrawBSPTree (BSP_tree *tree, point eye)
{
        if (tree == NULL) return;
        real result = tree->partition.ClassifyPoint (eye);
        if (result > 0)
        {
                DrawBSPTree(tree->back, eye);
                tree->polygons.DrawPolygons();
                DrawBSPTree(tree->front, eye);
        }
        else if (result < 0)
        {
                DrawBSPTree(tree->front, eye);
                tree->polygons.DrawPolygonList();
                DrawBSPTree (tree->back, eye);
        }
        else
        {
                // the eye point is on the partition plane...
                DrawBSPTree(tree->front, eye);
                DrawBSPTree(tree->back, eye);
        }
}
```

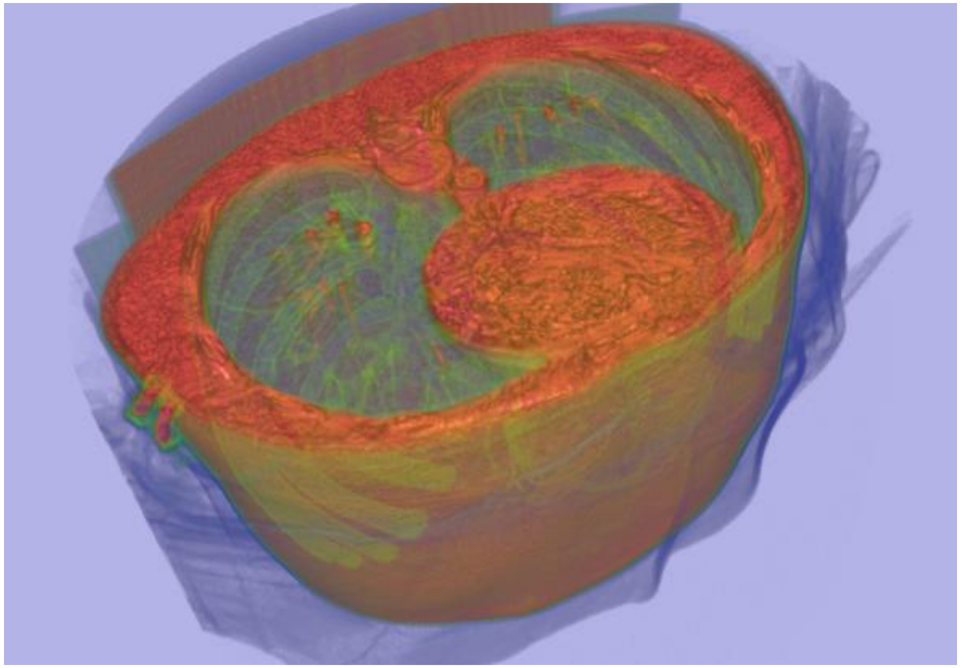
Visibility determination

- Combination of several algorithms
- Bbackface culling
- Frustum culling
- Pixel rewriting in color buffer when rendering
  - Rendering in front to back order
  - Structure in screen space for remembering which pixels were already filled – using 2D BSP tree

Transparency

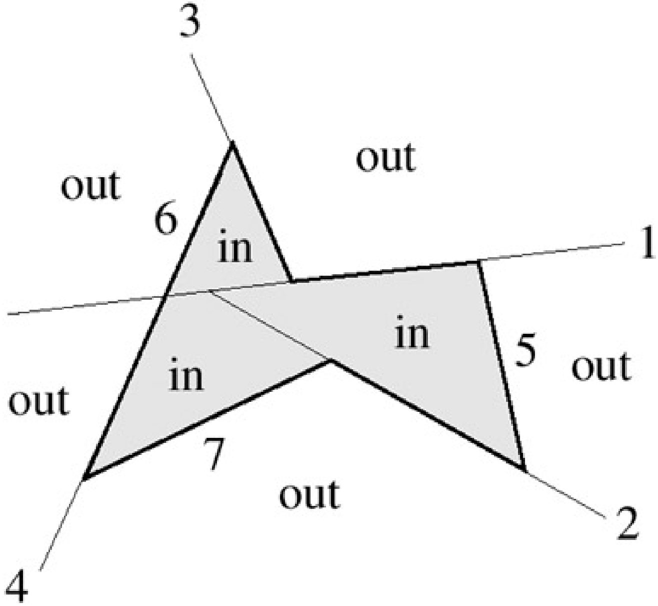
- Using blending (alpha-blending) in 3D
  - Fragments of currently rendered polygon are blended with color in framebuffer with some ratio
- Ordering of rendered polygons is needed
  - Front-to-back order
  - Back-to-front order
- Additive blending

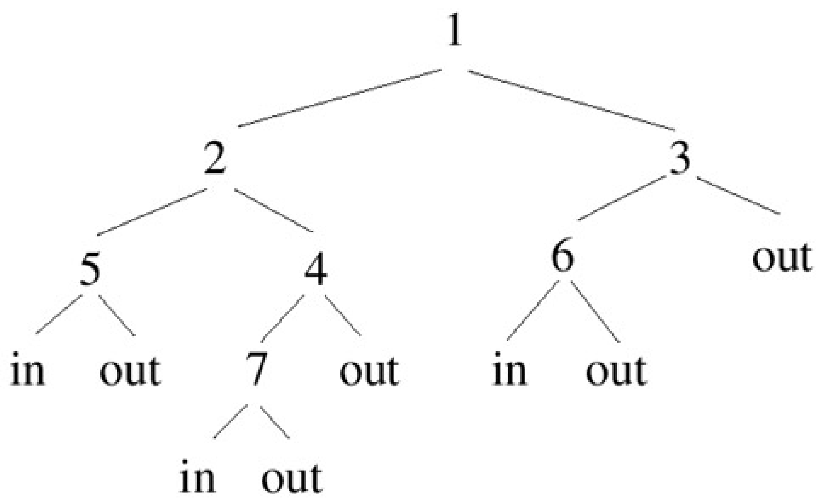




Objects representation

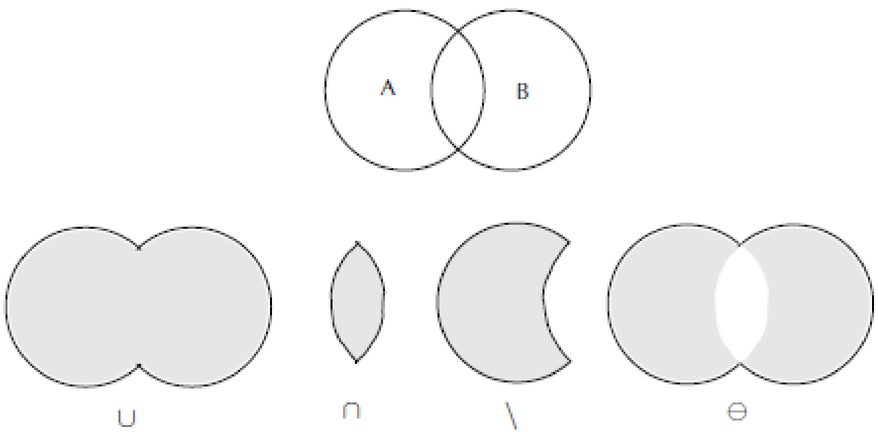
- Closed objects
- Border of objects defines subdivision hyperplanes
- Representation used for point test
- Unappropriate for smooth surfaces





Set operations on objects

- Crucial operations in geometric modeling

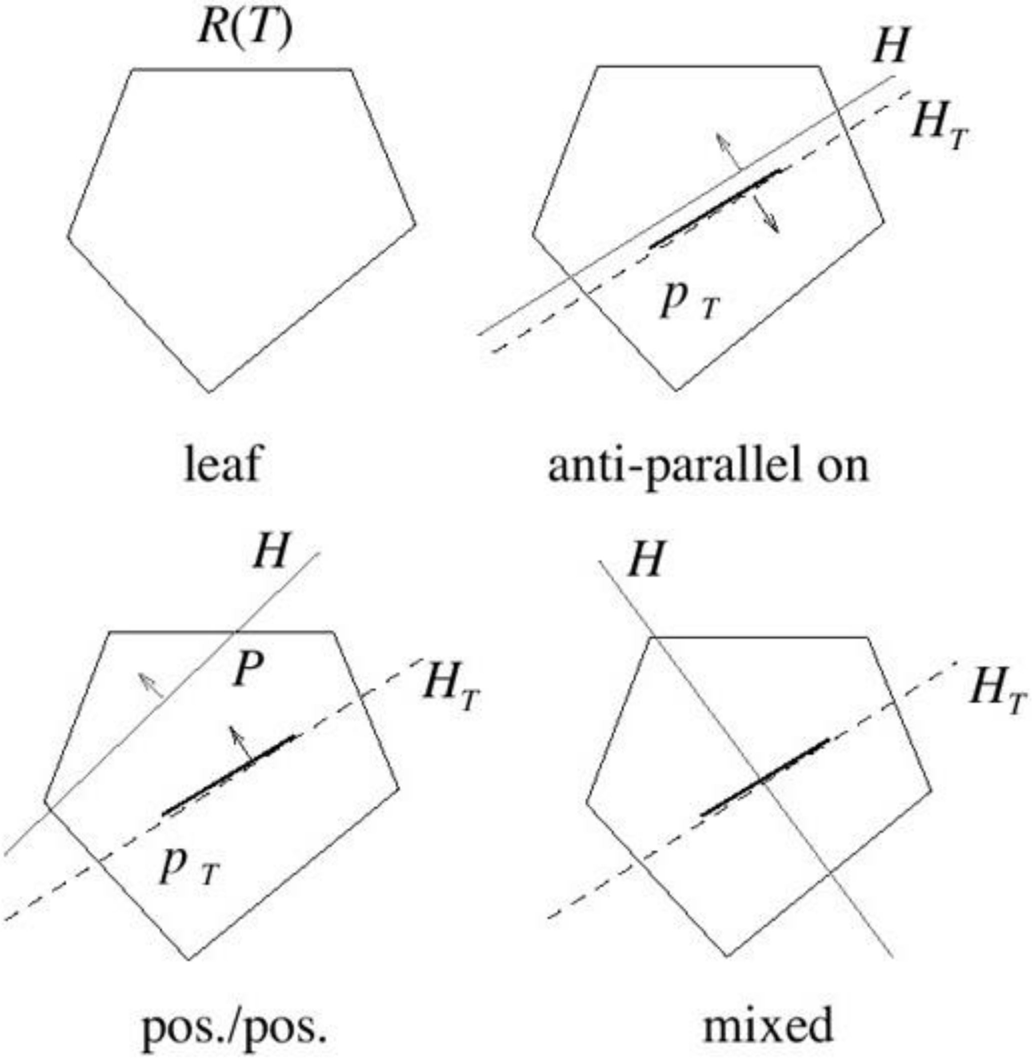


- BSP tree representation – connecting two BSP trees
- Union, intersection, difference – in BSP representation, difference only in elementary leaf operations

1. Part – BSP tree split

- For given BSP tree *T* and hyperplane *H*, create new BSP tree T_1, such that $T_1^- = T \cap H^-$ a $T_1^+ = T \cap H^+$
- *H* will be new root
- Node *T* consists of (H_T, p_T, T^-, T^+)
  - *H id* split hyperplane
  - *p* is polygon inside *H*
- Several configurations for hyperplane *H* in node *T* based on relative position of H and hyperplane in T
- Bounding volumes of each BSP tree node are needed

1. Part – BSP tree split



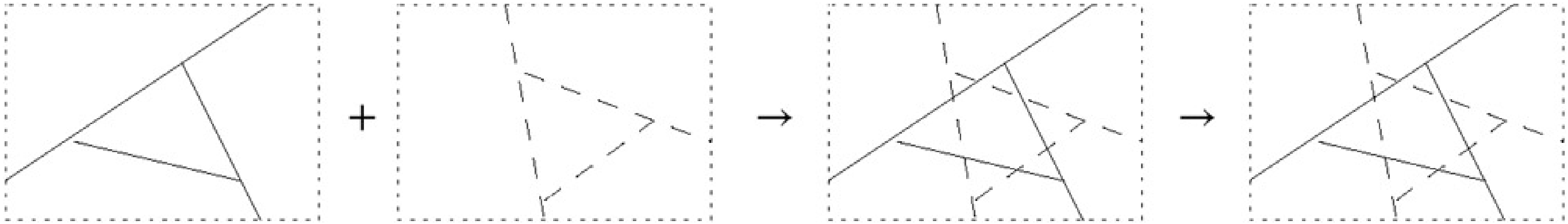

```
split-tree(T, H, P)
{
   //{P = H ∩ R(T)}
   // R(T) – region of the cell of node T (it is convex)
   case T is a leaf :
      return (H, T, T);
   case "anti-parallel" and "on" :
      return (H, T^+,T^-)
   case "pos./pos." :
      (T^{+1}, T^{+2}) = split-tree(T^+, H, P);
      T^1 = (H_T, p_T, T^-, T^{+1});
      T^2 = T^{+2};
      return (H, T^1, T^2);
   case "mixed" :
      (T^{+1}, T^{+2}) = split-tree(T^+, H, P ∩ R(T^+));
      (T^{-1}, T^{-2}) = split-tree(T^-, H, P ∩ R(T^-));
      T^1 = (H_T, p_T ∩ H^-, T^{-1}, T^{+1});
      T^2 = (H_T, p_T ∩ H^+, T^{-2}, T^{+2});
   return (H, T^1, T^2);
}
```

+ analogic cases

2. Part – BSP trees connection

- For given 2 BSP trees, concatenate it into one by inserting hyperplanes from first inside second
- If C_i are sets of elementary cell of i-th tree (represented by leafs of trees), then resulting tree T_3 has leaf cells:

$$C_3 = \{c_1 \cap c_2 | c_1 \in C_1\ c_2 \in C_2, c_1 \cap c_2 \neq \emptyset\}$$

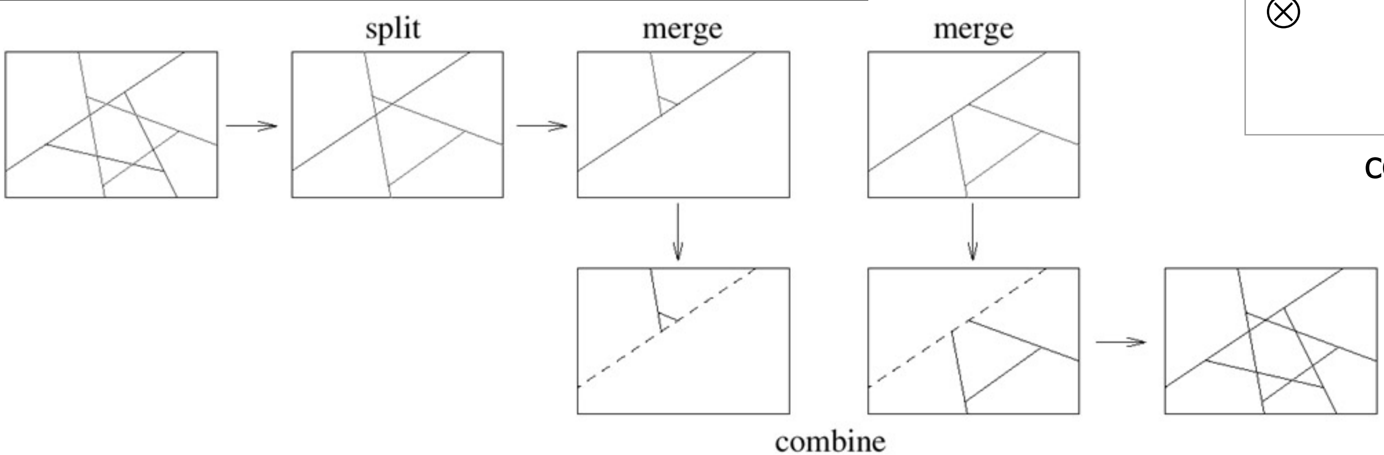


2. Part – BSP trees connection

```
merge(T1, T2) → T3
{
        if (T1 or T2 is a leaf)
        {
                perform the cell-op as required by the Boolean
                operation to be constructed
        }
        else
        {
                (T2+, T2-) = split-tree(T2, H1, …);
                T3- = merge (T1-, T2-);
                T3+ = merge (T1+, T2+);
                T3 = (H1, T3-, T3+);
                return T3;
        }
}
```

Operation	T_1	Result
$\cup$	in	T_1
	out	T_2
$\cap$	in	T_2
	out	T_1
$\setminus$	in	T_2^c
	out	T_1
$\otimes$	in	T_2^c
	out	T_2

cell-op, T_1 is leaf



Collision detection

- Checking intersection between nodes of two BSP trees
- Similar to raytracing algorithm
- Computation of hyperplanes intersections between cells
- When checking for collision of camera and object, computing intersection of segment and BSP tree

Shadow volumes

- BSP tree storing polygons of shadow volume
- Determination if given surface point is inside shadow volume = is in shadow

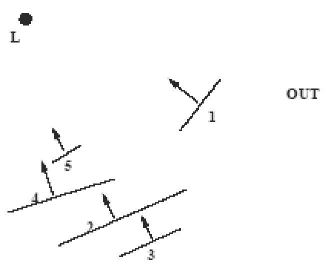


Figure 3: Initial scene

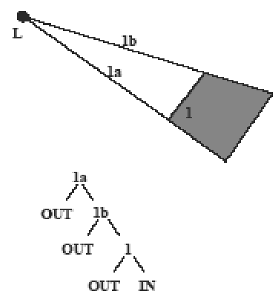


Figure 4: Insert poly 1

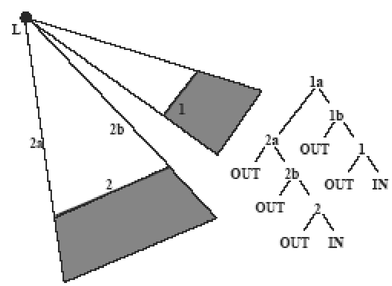


Figure 5: Insert poly 2

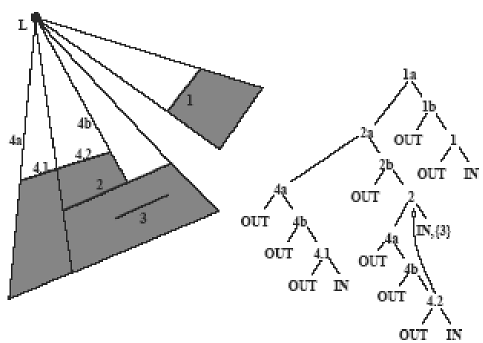


Figure 6: Insert poly 3 and 4

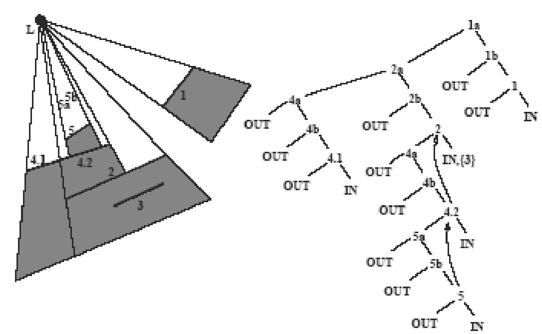


Figure 7: Insert poly 5

Shadow volumes

- Algorithm
  - From light position, find all silhouette edges of objects casting shadows
  - Each silhouette edge expand in the direction of light, creating polygons of shadow volumes
  - Create BSP tree for boundary polygons of shadow volumes
  - For any point in scene, find leaf node where it is positioned and read shadow information
  - Can be used stencil buffer instead of BSP tree

Dynamic scenes

- Dynamic objects are reinserted into BSP tree each frame
- Usually dynamic objects are represented as points and rendered before static objects
- Inserting one point is much faster than whole object with all boundary polygons
- Another option is to insert hyperplane perpendicular to view direction

Questions?